

## CODE DEFINITIONS

Although the 8K versions of Max-FORTH do not have an included assembler, code definitions can still be added. Three key words, CODE , END-CODE and CODE-SUB have been added to facilitate this.

CODE is a defining word that creates a head for the name string that follows it. The Code Field of the created word points to its own Parameter Field. This causes control to transfer to the code in the Parameter Field when the word is executed. CODE does not set the compilation mode but does leave security values on the stack for END-CODE , which ends the code definition.

Usually, in systems with assemblers, what goes between the CODE <name> and END-CODE, is assembly mnemonics and their parameters. Although there is an ASSEMBLER vocabulary, as required by the Assembler Extension Word Set, there was not enough room for a full assembler, so the vocabulary is empty. In order to enter machine code, the code must be hand assembled and placed in the dictionary using , and C, . At the end of the code definition's execution, control must be transferred back to the Max-FORTH system. This will usually be done by a JMP NEXT instruction. Of course, not knowing the address of NEXT can be a big drawback to using machine code definitions.

For that very reason, the defining word, CODE-SUB , was added to Max-FORTH. It allows the entry of a machine coded subroutine to be entered as a FORTH definition. Like CODE , CODE-SUB creates a head for its name string, and leaves security for END-CODE . CODE-SUB , however, puts the address of an interpreter in the new definition's Code Field. The interpreter will JSR to the new definition's Parameter Field when it is called, and will JMP NEXT after the called routine returns. The normal way to finish a CODE-SUB definition, is by compiling a RTS and using the standard END-CODE definition termination. The CODE-SUB definition can either be executed by name from high level FORTH or called from a machine code routine by JSR'ing to its Parameter Field.

The following example illustrates the creation of a code definition that causes the processor to go into the wait mode. It takes advantage of the previously described CODE-SUB word.

```
CODE-SUB WAIT
  3E C, ( WAIT INSTRUCTION )
  39 C, ( RTS )
END-CODE
```

It should be noted certain registers have significance to the virtual FORTH machine and require special treatment in code definitions. The Y register is used for the Data Stack Pointer. It should be left alone for the most part. (Note: the Data Stack grows down in memory.) If something is to be removed from the stack, it should be removed and THEN the Y register should be incremented twice (ie: LDD 0,Y INY INY). If something is to be put on the stack, the Y register should FIRST be decremented twice and then the value

should be put on (ie: DEY DEY STD 0,Y). If this order is not observed, the code produced may not be preemptable, and could cause high-level interrupt and multitasking failures.

The SP register is used for the FORTH Return Stack Pointer. No values can be taken from, or left on, the machine stack by the end of the definition. While a value can be temporarily pushed on the stack, it must come off before the return to NEXT.

The A, B, D, X and CC registers can be used without consequence.

Since the address of NEXT varies from version to version and revision to revision, a CODE-SUB is again used in the next example. Using a CODE definition, in the following example, would require providing the address of NEXT for all versions and revisions. Then, if a new revision was released with a different NEXT, this example might not work. Using the CODE-SUB defining word here, assures that this example will be able to return control to FORTH easily in all versions and revisions.

The example reads the first four A/D channels and places them on the stack.

```
CODE-SUB READ-A/D-CH0-3
  CE C, B030 , ( LDX $B030 )
  ( SET ADCTL FOR MULT READINGS, STRT CONV )
  86 C, 10 C, ( LDAA # 10 )
  A7 C, 00 C, ( STAA 0,X , $B030 )
  ( WAIT UNTIL CCF SET )
( SPIN ) 1F C, 00 C, 80 C, FC C, ( BRCLR 0,80,SPIN )
  4F C, ( CLRA )
  ( TAKE DATA, OPEN STACK, STORE DATA )
  E6 C, 01 C, ( LDAB 1,X )
  18 C, 09 C, ( DEY )
  18 C, 09 C, ( DEY )
  18 C, ED C, 00 C, ( STD 0,Y )
  E6 C, 02 C, ( LDAB 2,X )
  18 C, 09 C, ( DEY )
  18 C, 09 C, ( DEY )
  18 C, ED C, 00 C, ( STD 0,Y )
  E6 C, 03 C, ( LDAB 3,X )
  18 C, 09 C, ( DEY )
  18 C, 09 C, ( DEY )
  18 C, ED C, 00 C, ( STD 0,Y )
  E6 C, 04 C, ( LDAB 4,X )
  18 C, 09 C, ( DEY )
  18 C, 09 C, ( DEY )
  18 C, ED C, 00 C, ( STD 0,Y )
  39 C, ( RTS )
END-CODE
```

One other requirement of the Assembler Extension Word Set is the ;CODE word. This word (in conjunction with a newly defined defining word) causes later defined words to use the machine code interpreter following the ;CODE . As an example the following program segment simulates the set up for a two engine monitoring program that uses identically names for both engine's parameters. A variable CHANNEL is used to control a new type variable created by the defining word CHANNEL-VARIABLE .

```
0 CONSTANT PORT
1 CONSTANT STBD
VARIABLE CHANNEL ( VALUE SHOULD BE 0 OR 1 )
```

```
: MAKE-CHANNEL-VARIABLE <BUILDS 0 , 0 ,
;CODE
FC C, ' CHANNEL @ , ( LDD CHANNEL )
F3 C, ' CHANNEL @ , ( ADDD CHANNEL )
8F C, ( XGDX )
EC C, 00 C, ( LDD 0,X )
18 C, 09 C, ( DEY )
18 C, 09 C, ( DEY )
18 C, ED C, ( STD 0,Y )
7E , FE4E , ( JMP VERSION 1.0 NEXT )
END-CODE
```

```
MAKE-CHANNEL-VARIABLE ENGINE-SUPPLY-TEMP
MAKE-CHANNEL-VARIABLE ENGINE-RETURN-TEMP
MAKE-CHANNEL-VARIABLE ENGINE-SUPPY-FLOW
MAKE-CHANNEL-VARIABLE ENGINE-RETURN-FLOW
MAKE-CHANNEL-VARIABLE ENGINE-RPM
```

```
PORT CHANNEL ! ( NOW WORK ON PORT ENGINE )
( etc. ... )
STBD CHANNEL ! ( NOW WORK ON STBD ENGINE )
( etc. ... )
```